MP Insem Answers - NOTES

Computer Networks (RMD engineering college)

Scan to open on Studocu

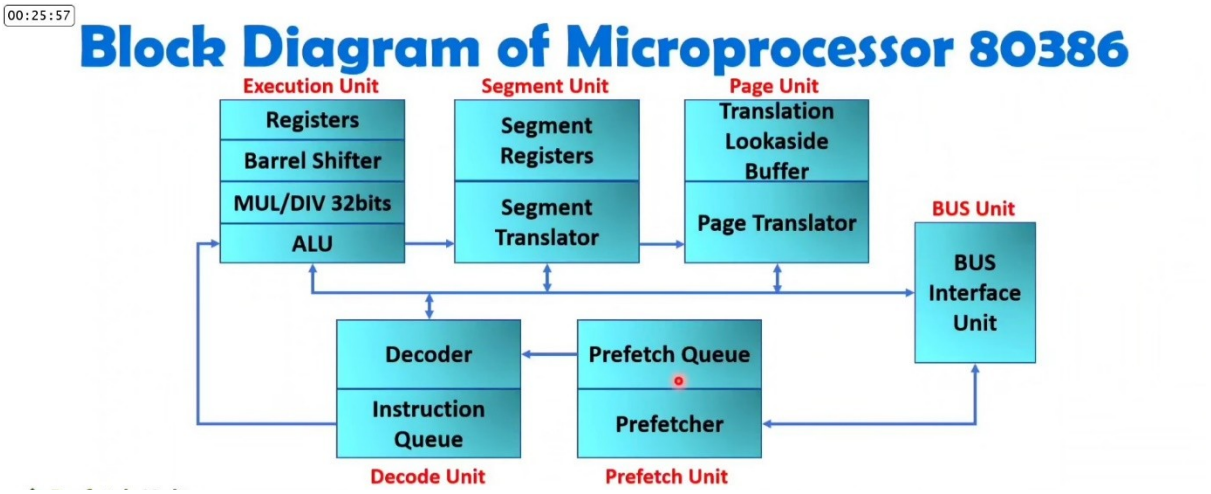
Studocu is not sponsored or endorsed by any college or university
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

In-Sem Answers

Q1)

a) Explain the architecture of the 80386 microprocessors, with an appropriate diagram.

→



Key Components:

1. **Execution Unit (EU):**
    - Contains the Arithmetic Logic Unit (ALU) visible in the block diagram, which performs arithmetic and logical operations on data (shown as **ALU**).
    - It also includes registers (not explicitly shown in the diagram) to hold data operands during these operations.
2. **Bus Interface Unit (BIU):**
    - Manages communication between the processor and external components like memory and I/O devices.
    - The Bus Interface Unit likely includes the block labeled **BUS Unit** in the diagram.
    - Address and data transfers on the address bus, data bus, and control lines are coordinated by this unit.
3. **Control Unit (CU):**

This document is available on studocu
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- Fetches instructions from memory using the Prefetch Unit (explained below).
- Decodes instructions and generates control signals to coordinate operations of the ALU, registers, and BIU. The block labeled **Decoder** in the diagram likely handles this functionality.

4. **Prefetch Unit:**
   - Fetches instructions from memory based on the program counter (PC) register.
   - Supplies instructions to the decode unit for processing. This functionality is likely handled by the block labeled **Prefetch Queue** in the diagram.
5. **Registers:**
   - The 80386 has a set of general-purpose registers, segment registers, and control registers. These registers are not shown explicitly in the block diagram, but they are essential components for storing data and control flags during program execution.

b) Describe the following addressing modes of 80386 with example,

i) Index addressing mode

ii) Direct addressing mode

iii) Based index mode

→ **Addressing Modes of the 80386:**

i) **Index Addressing Mode:**

- **Description:** In index addressing mode, the effective address of the operand is calculated by adding an index value to a base address.
- **Example:** MOV EAX, [ESI + 100]

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- In this example, the contents of the ESI register (base address) are added to the constant value 100 (index) to calculate the effective address. The data stored at this effective address is then moved into the EAX register.

ii) **Direct Addressing Mode:**

- **Description:** In direct addressing mode, the operand directly specifies the memory location where the data is stored.
- **Example:** MOV EBX, [0xABCD]
  - Here, the data stored at memory location 0xABCD is moved into the EBX register.

iii) **Based Index Mode:**

- **Description:** Based index addressing mode combines elements of both index and base addressing modes. It involves adding an index value to a base address and then using the result as the address to access memory.
- **Example:** MOV EDX, [EBX + ECX*4]
  - In this example, the contents of the EBX register (base address) are added to four times the value in the ECX register (index) to calculate the effective address. The data stored at this effective address is then moved into the EDX register.

c) Describe the use of various 80386 Data Movement Instructions in assembly language programming with examples.

→

1. **MOV (Move):**
   - **Description:** The MOV instruction is used to move data from a source operand to a destination operand. It can transfer data between registers, memory locations, and immediate values.

This document is available on studocu

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- **Example:** MOV AX, BX copies the contents of register BX to register AX.

2. **XCHG (Exchange):**
    - **Description:** The XCHG instruction swaps the contents of two operands. It can exchange values between registers or between a register and a memory location.
    - **Example:** XCHG AX, BX swaps the values in registers AX and BX.
3. **PUSH (Push onto Stack):**
    - **Description:** The PUSH instruction pushes the source operand onto the top of the stack. It decrements the stack pointer and stores the value in the memory location pointed to by the stack pointer.
    - **Example:** PUSH AX pushes the value in register AX onto the stack.
4. **POP (Pop from Stack):**
    - **Description:** The POP instruction retrieves the value from the top of the stack and stores it in the destination operand. It increments the stack pointer.
    - **Example:** POP BX pops the value from the stack and stores it in register BX.
5. **CWD (Convert Word to Doubleword):**
    - **Description:** The CWD instruction is used to sign-extend a 16-bit value in the AX register to a 32-bit value. It copies the sign (bit 15) of AX into every bit of the DX register.
    - **Example:** CWD extends the sign of a 16-bit value in AX to a 32-bit value in DX:AX.
6. **MOVSX (Move with Sign-Extension):**
    - **Description:** The MOVSX instruction is used to copy a value from a source operand to a destination operand with sign-

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

extension. It is often used when converting a smaller signed value to a larger size.

Example: MOVSX EAX, AL copies the value from AL (8-bit) to EAX (32-bit) with sign-extension.

Q2)

a) Describe the various operating modes of 80386.

→

The 80386 microprocessor supports different operating modes to provide compatibility with earlier processors and introduce advanced features. The primary operating modes of the 80386 are:

1. **Real Mode:**
    - In Real Mode, the 80386 behaves like its predecessor, the 80286, and is backward compatible with the 8086/8088 processors.
    - It operates with a segmented memory model, allowing access to only 1 MB of physical memory.
    - The processor uses 16-bit addressing, and all instructions from the 8086/8088 can be executed in this mode.
2. **Protected Mode:**
    - Protected Mode is a more advanced operating mode of the 80386, offering features like multitasking, memory protection, and virtual memory.
    - It supports a flat 32-bit addressing model, allowing access to a theoretical 4 GB of physical memory.
    - Memory protection features prevent one program from interfering with another, enhancing system stability.
    - Multitasking is facilitated by providing each task with its own protected memory space.
3. **Virtual 8086 Mode:**

This document is available on studocu

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- Virtual 8086 Mode allows the execution of multiple 8086 virtual machines within the Protected Mode environment.
- This mode is designed to maintain compatibility with older software that was written for the 8086 processor.
- Each virtual machine has its own 1 MB memory space and operates as if it were running on a separate 8086 processor.

4. **System Management Mode (SMM):**
    - SMM is a special-purpose operating mode designed for power management and system maintenance tasks.
    - It allows the processor to enter a low-power state when the system is idle, contributing to energy efficiency.
    - SMM is often used for handling events like power button presses and managing system-level functions independently of the operating system.

b) Describe the following addressing modes of 80386 with example

i) Register addressing mode

ii) Register Indirect addressing mode

iii) Immediate addressing mode

→ **Addressing Modes of the 80386:**

i) **Register Addressing Mode:**

- **Description:** In register addressing mode, the operand is directly specified by a register.
- **Example:** MOV AX, BX
    - In this example, the data in register BX is moved to register AX.

ii) **Register Indirect Addressing Mode:**

- **Description:** In register indirect addressing mode, the operand is a memory address stored in a register. The data at that memory address is accessed.

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- **Example:** MOV AX, [BX]
    - Here, the contents of the memory location whose address is stored in register BX are moved to register AX.

iii) **Immediate Addressing Mode:**

- **Description:** In immediate addressing mode, the operand is a constant value specified directly in the instruction itself.
- **Example:** MOV AX, 1000h
    - In this example, the immediate value 1000h is moved into register AX.

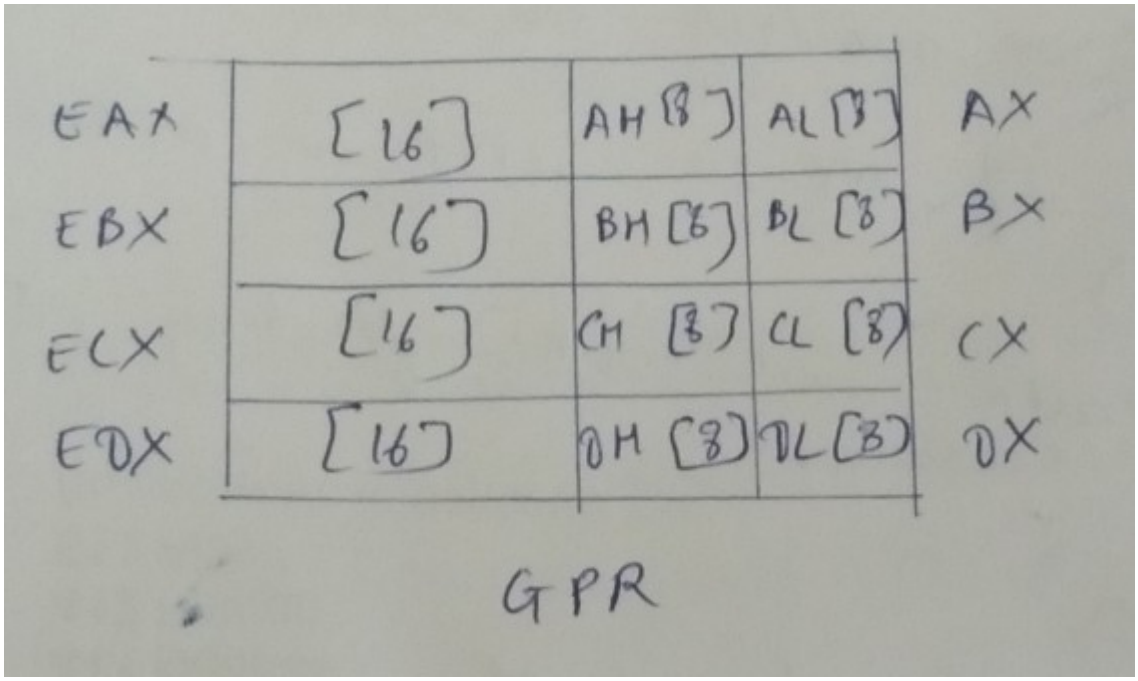
c) Explain the General Registers and Segment Registers of 80386 with an appropriate diagram.

→

1. **General Registers:**
    - The 80386 has eight general-purpose registers, namely EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP. Each register is 32 bits wide.
    - These registers are used for general data manipulation, arithmetic, and logical operations.
    - They can also be used as pointers for addressing memory locations and passing function arguments.

This document is available on studocu

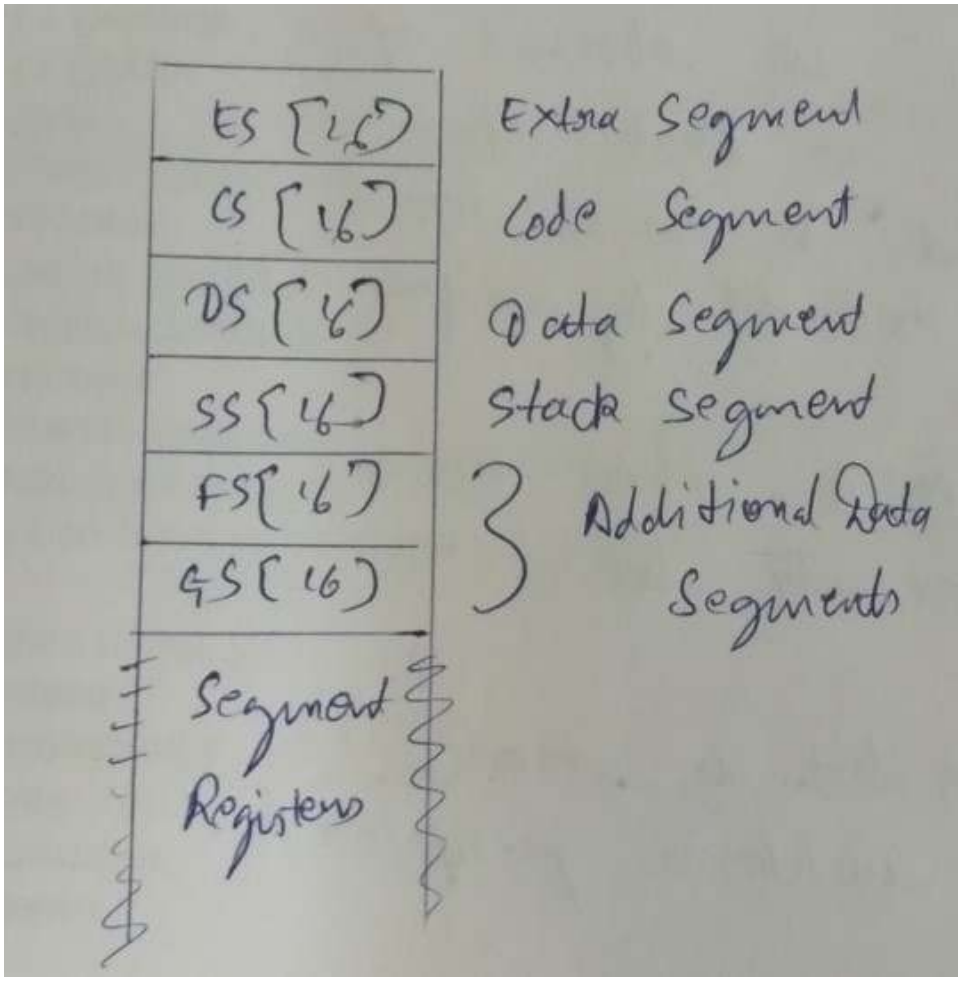
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)



Segment Registers:

- The 80386 has six segment registers, namely CS, DS, SS, ES, FS, and GS.
- These registers are used to store segment selectors, which are used for memory segmentation.
- CS (Code Segment) points to the segment containing the currently executing code.
- DS (Data Segment) points to the segment containing the data accessed by default data movement instructions.
- SS (Stack Segment) points to the segment containing the stack.
- ES (Extra Segment) is an additional data segment register that can be used for certain data operations.
- FS and GS are additional segment registers introduced in the 80386 architecture, providing extended addressing capabilities.

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)



Q3)

a) Explain the 80386 processor state after Reset.

→ After a reset, the 80386 processor is initialized to a specific state to ensure proper operation. Here's an explanation of the processor state after reset:

1. **Operating Mode:**
    - The processor is initially in Real Mode after a reset. Real Mode is a backward-compatible mode with earlier x86 processors like the 8086/8088. In Real Mode, the processor operates with a 20-bit address bus, allowing it to address up to 1 MB of memory.
2. **Segment Registers:**

This document is available on studocu
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- The segment registers CS, DS, SS, and ES are set to 0x0000, pointing to the bottom of the physical memory. In Real Mode, these segment registers are 16 bits wide.

3. **Instruction Pointer (IP):**
   - The instruction pointer (IP) is set to 0xFFFF, indicating the initial address from which the processor fetches instructions. In Real Mode, the IP register is 16 bits wide.
4. **Control Registers:**
   - Control registers like CR0 are initialized with default values suitable for Real Mode operation. For example, the PE (Protection Enable) bit in CR0 is cleared to disable memory protection.
5. **Status Registers:**
   - Status registers like the FLAGS register are cleared to their default state, with all flags set to 0.
6. **Stack Pointer (SP):**
   - The stack pointer (SP) is set to the top of the system stack, which typically starts at the top of the physical memory and grows downward.
7. **Processor Cache:**
   - The processor cache is typically invalidated, ensuring that the processor fetches instructions and data from memory after reset.

b) Draw and Explain Read Cycle with non-pipelined address timing.

→

The 80386 read cycle with non-pipelined address timing involves a series of steps to fetch data from memory:

1. **Address Output (T1):**

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- The processor places the memory address of the data it wants to read onto the address bus.
- Control signals on the control bus indicate a read operation.
- This initial state lasts for one processor clock cycle (T1).

2. **Memory Access (Variable T2):**
   - The memory interface unit receives the address and starts the memory access process.
   - The time it takes for memory to retrieve the data can vary depending on memory speed and access time.
   - This period is referred to as **T2** and can last for multiple processor clock cycles.

This document is available on studocu

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

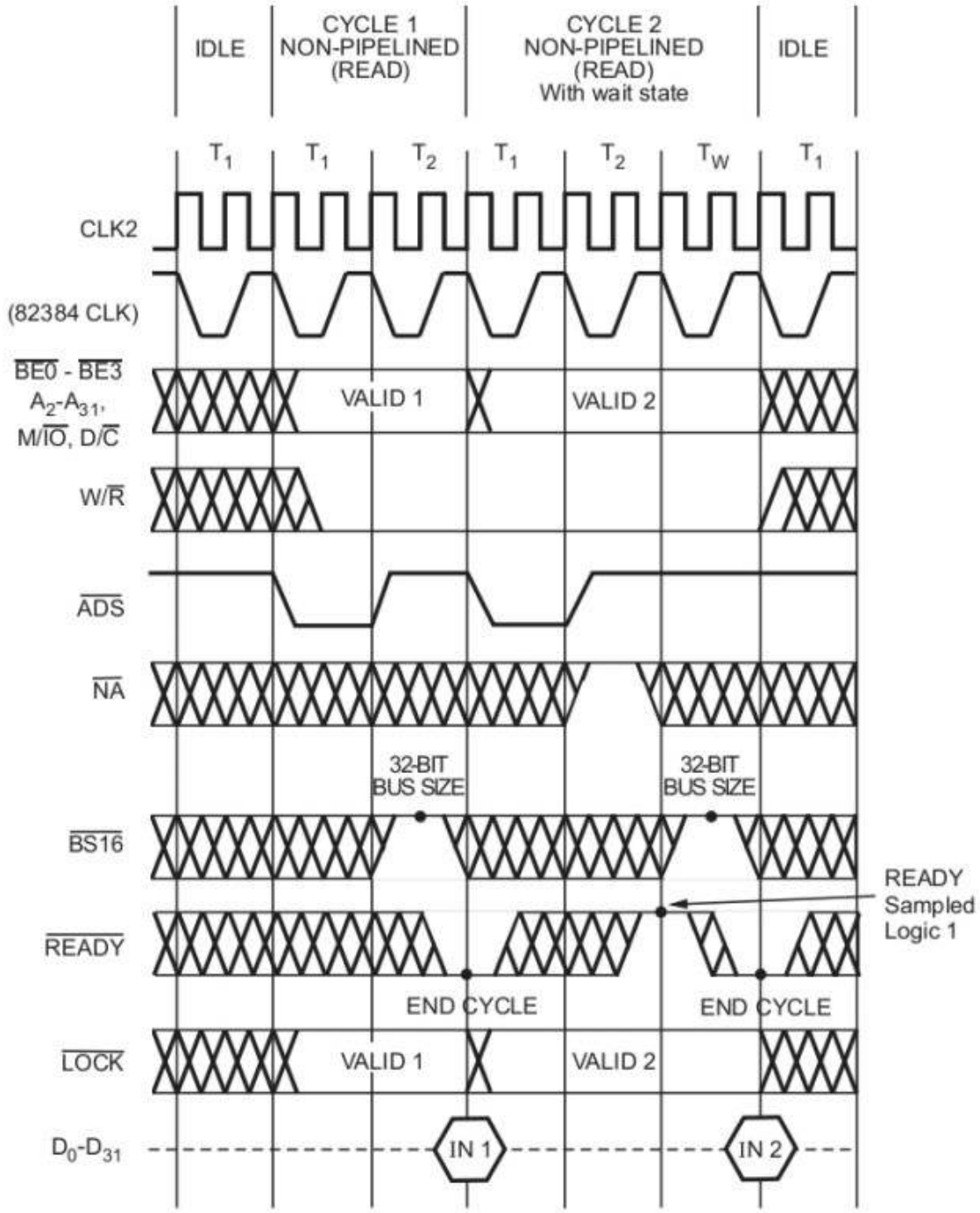


Fig. 3.6.5 Non pipelined read cycle

3. **Ready Check (Variable):**
    - The processor doesn't proceed immediately after T1. It enters a wait state and repeatedly checks a signal called **READY** on the control bus.
    - The memory asserts the READY signal when the data is ready to be transferred.

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

4. **Data Transfer (T3):**
    - Once the memory asserts READY, signaling that the data is available, the read cycle proceeds.
    - The memory places the data on the data bus.
    - The processor reads the data from the data bus and stores it in the designated register.
    - This data transfer typically takes one processor clock cycle (T3).
5. **Cycle Termination:**
    - After data transfer, the read cycle is complete.
    - The processor and memory interface can initiate the next operation.

c) Draw and Explain Control Registers of 80386.

→ **CR0:**

The lower 16-bits of CR0 are also called as MSW (Machine Status Word), for compatibility with 80286.
It contains six status bits: PG (Paging Enable), ET (Extension Type), TS (Task Switched), EM (Emulation), MP (Monitor Coprocessor), PE (Protection Enable).

1. **PG – Paging Enable:** When PG=1, paging is enabled. When PG=0, paging is disabled.

2. **ET – Extension Type:** The ET bit indicates if 80387 is connected or a previous math coprocessor is connected.

3. **TS – Task Switched:** The processor sets this bit whenever a task switch operation is performed.

4. **EM – Emulation:** When EM=1, it causes INT 11 on a floating-point operation where the floating-point instruction may be emulated, when the processor does not have a floating-point unit.

This document is available on studocu
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

1. There are four control registers (CR0, CR1, CR2, CR3).
2. Only four of them are used by the current implementation; register CR1 is reserved for future use.

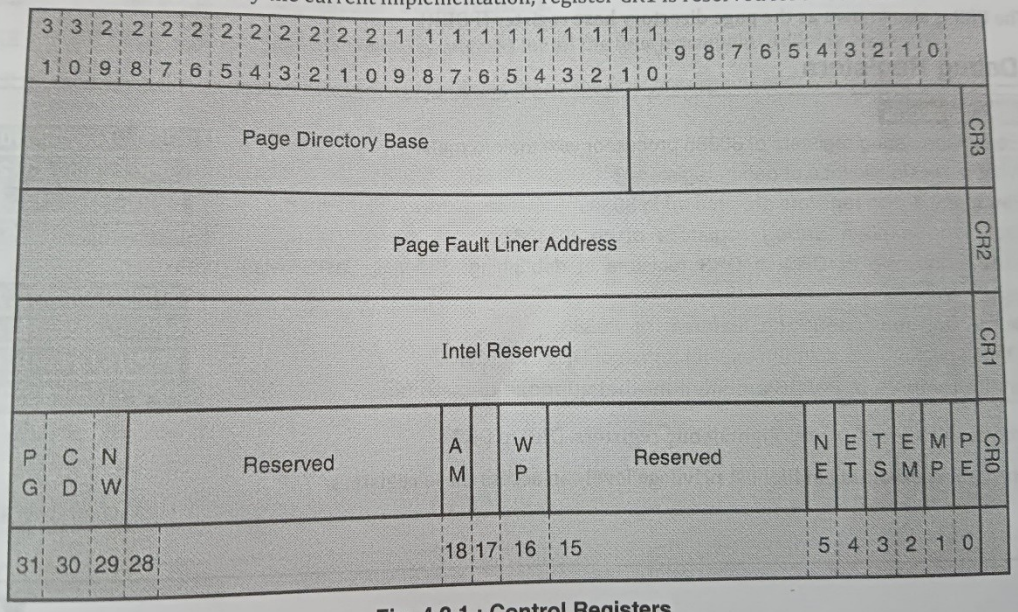


Fig. 4.2.1 : Control Registers

5. **MP – Monitor Coprocessor:** On the i286 and i386 processors, the MP bit controls the function of the WAIT instruction, which makes the processor to wait for the completion of the task by math coprocessor. When it is '1', it indicates a coprocessor is connected and on wait instruction the host processor has to wait for the TEST# pin to be enabled.

6. **PE – Protection Enable:** When the processor enters into the protected mode it makes PE=1, and the protection mechanism is enabled. When PE=0, the processor operates in real (fast 8086) mode.

CR1:

It is reserved by Intel for future use.

CR2:

The CR2 register holds the 32-bit linear address that caused the last page fault detected.

CR3:

The CR3 is also known as the page directory base register (PDBR).

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

Q4)

a) Explain the following signals

1. INTR

2. ADS#

3. READY#

4. HOLD

5. NMI

→

1. **INTR (Interrupt Request):**
    - **Description:** INTR is an active-high signal used by external devices to request an interrupt. When an external device asserts the INTR signal, it indicates to the processor that it needs attention and wishes to interrupt the current execution flow.
    - **Use:** The processor responds to an INTR request by completing the current instruction, storing the current execution context, and jumping to the interrupt service routine (ISR) to handle the interrupt.
2. **ADS# (Address Status):**
    - **Description:** ADS# is an active-low signal used in microprocessor systems to indicate the validity of the address bus during the first clock cycle of a bus transaction.
    - **Use:** When ADS# is low, it indicates that the address bus contains a valid address. It is typically used to latch the address on the bus by memory or other peripheral devices.
3. **READY# (Ready):**
    - **Description:** READY# is an active-low signal used by memory and peripheral devices to indicate their readiness to respond to a read or write operation from the microprocessor.

This document is available on studocu

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- **Use:** When READY# is low, it indicates that the memory or peripheral device is ready to respond to the microprocessor's request. The microprocessor waits for READY# to become active before proceeding with the data transfer.

4. **HOLD (Hold):**
    - **Description:** HOLD is an input signal used by external devices to request control of the system bus from the microprocessor temporarily.
    - **Use:** When an external device asserts the HOLD signal, it requests the microprocessor to relinquish control of the bus. The microprocessor responds by entering a hold state, suspending its current operation until the HOLD signal is deasserted.
5. **NMI (Non-Maskable Interrupt):**
    - **Description:** NMI is an active-high signal used to request a non-maskable interrupt, which is an interrupt that cannot be disabled or masked by the processor.
    - **Use:** When an external device asserts the NMI signal, it triggers a non-maskable interrupt, causing the processor to immediately stop its current operation and jump to a predefined NMI handler routine. NMIs are typically used for critical system events that require immediate attention, such as hardware faults or system resets.

b) Draw and explain the Write Cycle with non-pipelined address timing.

→ The 80386 write cycle with non-pipelined address timing involves a series of steps to store data into memory:

1. **Address and Control Output (T1):**
    - The processor places the memory address of the destination location onto the address bus.
    - Control signals on the control bus indicate a write operation.
    - This initial state lasts for one processor clock cycle (T1).

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

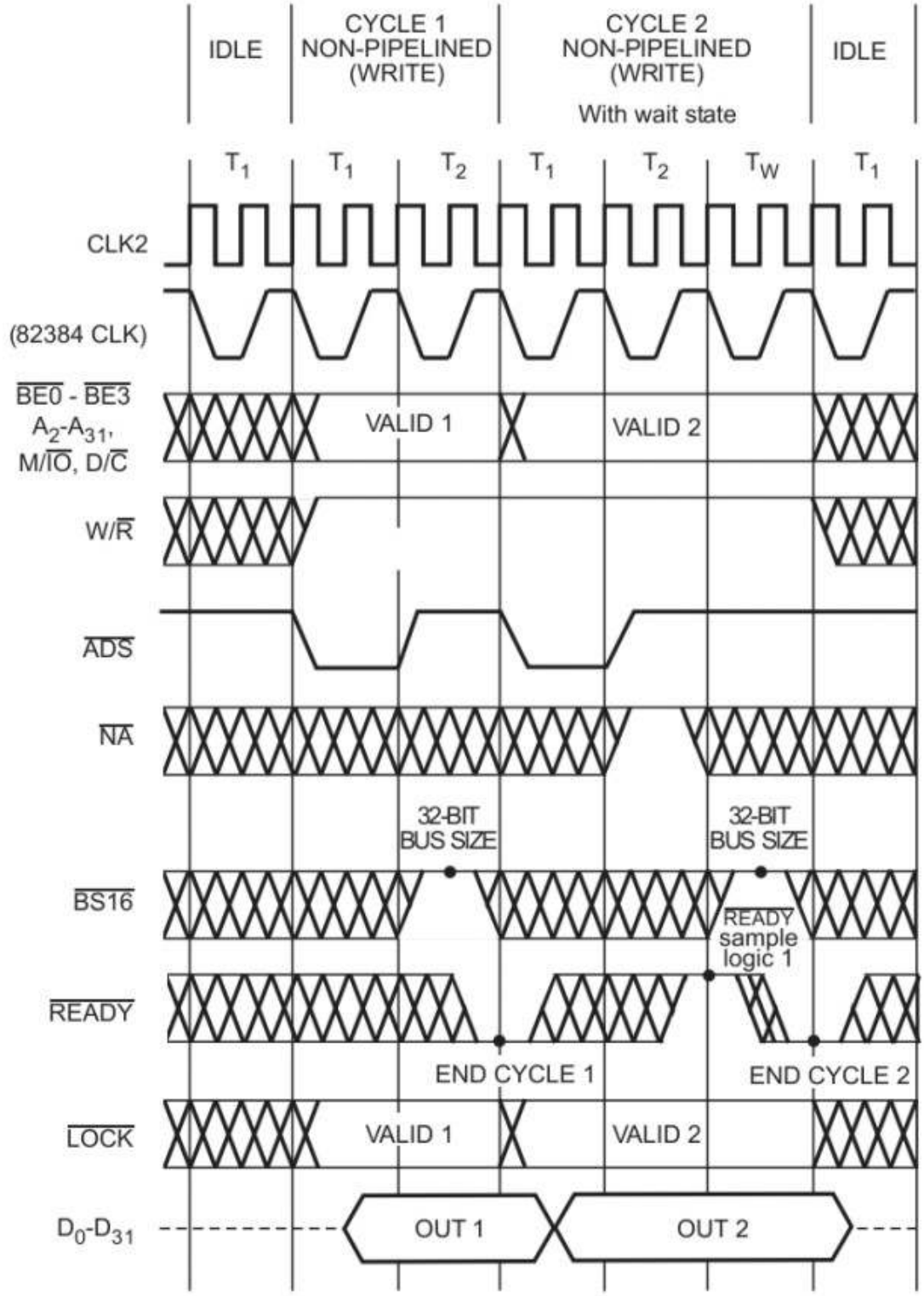


Fig. 3.6.6 Non-pipelined write cycle

2. **Data Output (Optional):**

This document is available on studocu

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- The processor may also place the data to be written onto the data bus in this cycle (T1) for some write operations.

3. **Memory Access Setup (T2):**
   - The memory interface unit receives the address and control signals.
   - It prepares the memory location for data reception.
   - This period (T2) can vary depending on memory speed and access time.
4. **Ready Check (Variable):**
   - Similar to the read cycle, the processor enters wait states after T1 or T2 (depending on data output timing).
   - It repeatedly checks the **READY** signal on the control bus from the memory.
5. **Data Transfer (T3):**
   - Once the memory asserts READY, signaling it's ready to receive data, the processor places the data on the data bus (if not done in T1).
   - The memory interface unit stores the data in the specified memory location.
   - This data transfer typically takes one processor clock cycle (T3).
6. **Cycle Termination:**
   - After data transfer (or after READY if data was already sent), the write cycle is complete.
   - The processor and memory interface can initiate the next operation.

c) Draw and Explain Debug Registers of 80386.

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

→

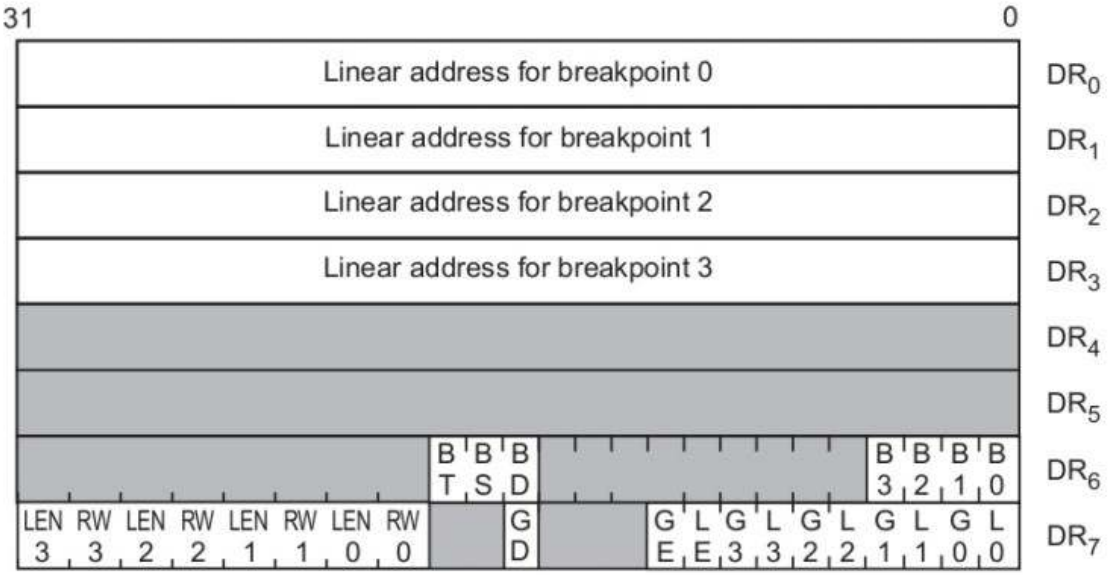


Fig. 3.8.3 Debug registers

The 80386 microprocessor features a set of debug registers that facilitate debugging and monitoring of program execution. These registers are essential for implementing breakpoints, watching memory access, and controlling debugging operations. Below is an explanation along with a diagram illustrating the debug registers of the 80386:

Debug Registers of 80386:

1. **DR0 to DR3:**
    - The 80386 microprocessor has four debug registers labeled as DR0, DR1, DR2, and DR3.
    - These registers are used to store linear addresses representing locations in memory where breakpoints can be set.
    - Each register is 32 bits wide, allowing the storage of a 32-bit linear address.
2. **DR6 (Debug Status Register):**
    - DR6 is the debug status register, also known as the debug control register 6.
    - It contains status bits that indicate the cause of debug exceptions, such as breakpoints or other debug events.

This document is available on studocu
Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)

- The bits in DR6 are used to determine which debug events have occurred and need attention.

3. **DR7 (Debug Control Register):**
    - DR7 is the debug control register, also known as the debug control register 7.
    - It is used to configure and control the behavior of debug exceptions and breakpoints.
    - DR7 allows the programmer to enable or disable breakpoints, specify the type of breakpoints (instruction or data), and control debug event reporting.

Downloaded by Tyler Silas (tylersilas.acadcommision@gmail.com)